Instytut Informatyki
Politechnika Śląska

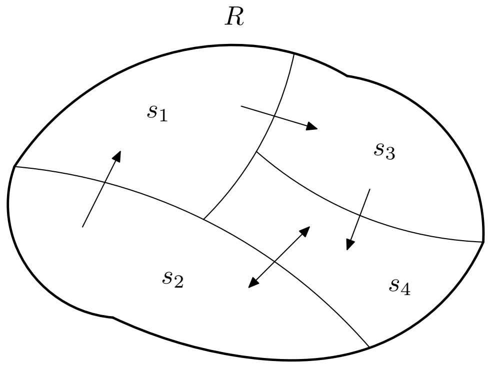
Obliczenia równoległe

Opracował: Zbigniew J. Czech

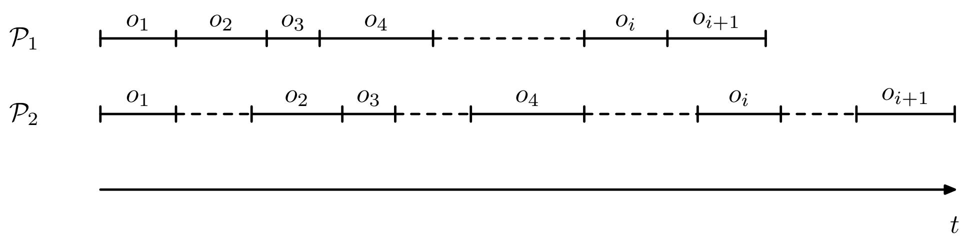
materiały dydaktyczne
Gliwice, luty 2016

Spis treści

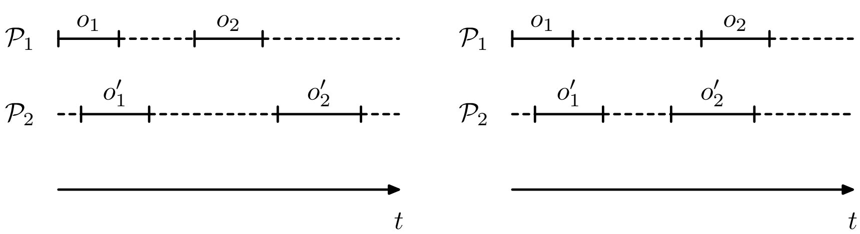
1 Procesy współbieżne



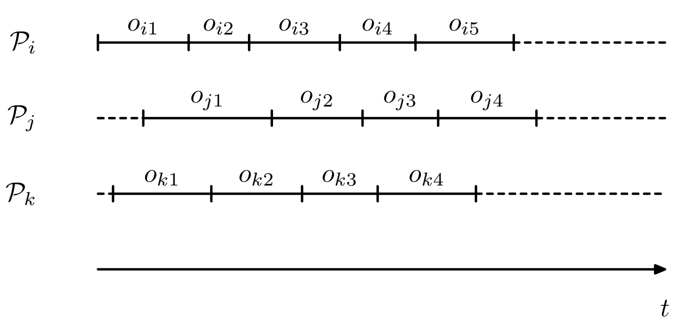
Rysunek 1: Algorytm równoległy R złożony z algorytmów składowych s_1, s_2, s_3 i s_4 rozwiązujących poszczególne zadania (podproblemy); strzałki oznaczają współpracę algorytmów składowych, tj. przesłania danych lub synchronizację działania



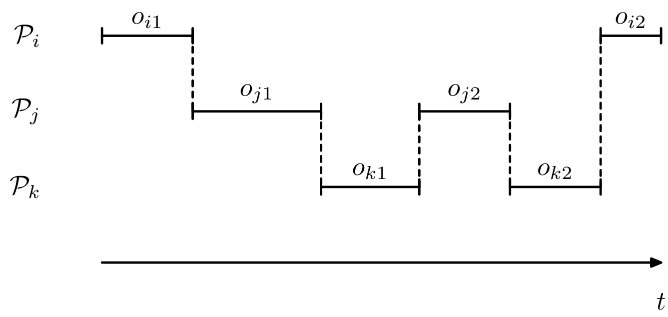
Rysunek 2: Procesy sekwencyjne $\mathcal{P}_1$ i $\mathcal{P}_2$ równoważne pod względem wyników obliczeń; t oznacza oś czasu



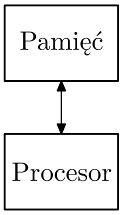
Rysunek 3: Dwie możliwe realizacje procesów $\mathcal{P}_1$ i $\mathcal{P}_2$



Rysunek 4: Jednoczesne wykonywanie operacji procesów $\mathcal{P}_i$, $\mathcal{P}_j$ oraz $\mathcal{P}_k$



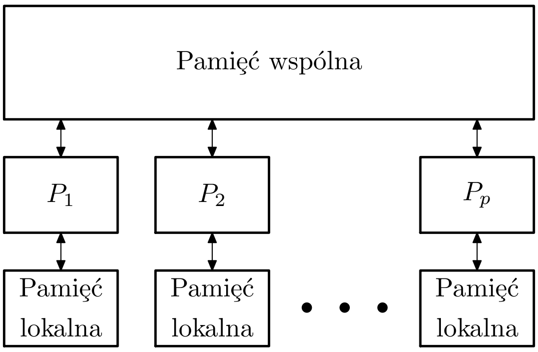
Rysunek 5: Przeplatanie operacji procesów — jeden procesor rzeczywisty „implementuje" trzy procesory wirtualne



Rysunek 6: Model obliczeń sekwencyjnych RAM

2 Podstawowe modele obliczeń równoległych

1. $M_4 := M_1 + 5$ {zapisanie w komórce M_4 sumy zawartości komórki M_1 i stałej 5}
2. $M_0 := M_{M_4}$ {adresowanie pośrednie: zapamiętanie w komórce M_0 zawartości}
   {komórki, której adres jest w komórce M_4}
3. $M_3 := M_1 - 1$ {obliczenie numeru iteracji; $M_3 = n - 1$}
4. $M_0 := M_0 * M_2$ {zapisanie w komórce M_0 iloczynu zawartości komórek M_0 i M_2}
5. $M_4 := M_3 + 5$ {obliczenie indeksu współczynnika wielomianu}
6. $M_0 := M_0 + M_{M_4}$
7. $M_3 := M_3 - 1$
8. **goto** 4 **if** $M_3 \geqslant 0$ {wykonanie instrukcji 4, jeśli zawartość komórki M_3 jest nieujemna}
9. **halt** {koniec obliczeń}

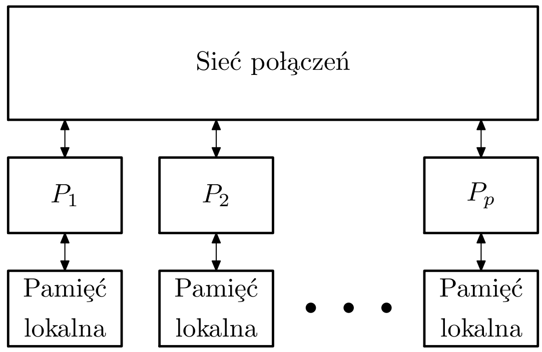


Rysunek 7: Model obliczeń równoległych PRAM

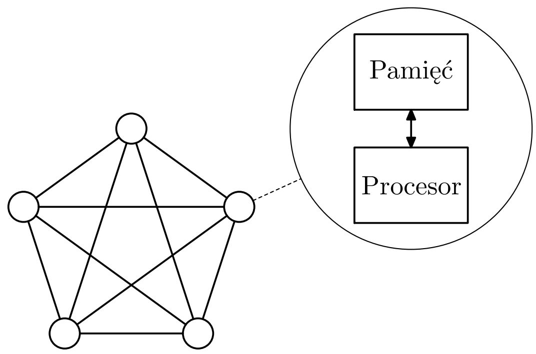
3 Elementarne algorytmy równoległe

3.1 Miara Karpa i Flatta

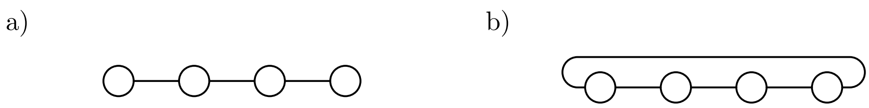
Patrz rys. 16, rys. 17 oraz tab. 2.



Rysunek 8: Model sieciowy obliczeń równoległych



Rysunek 9: Sieć pełna

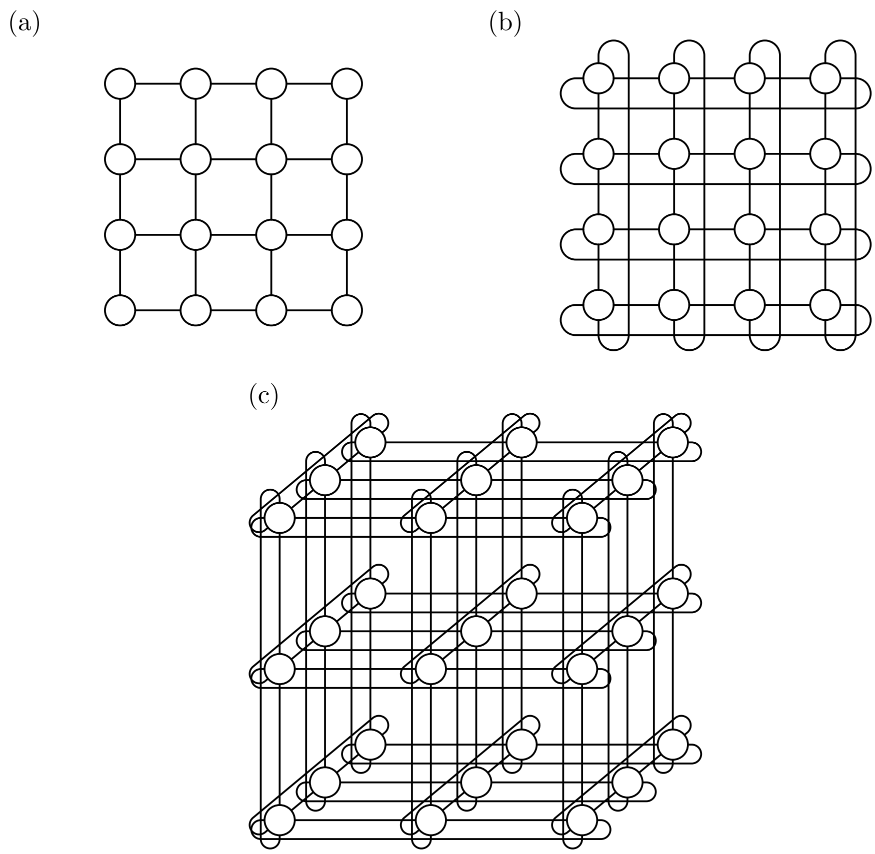


Rysunek 10: a) Jednowymiarowa siatka oraz b) jednowymiarowy torus (pierścień)

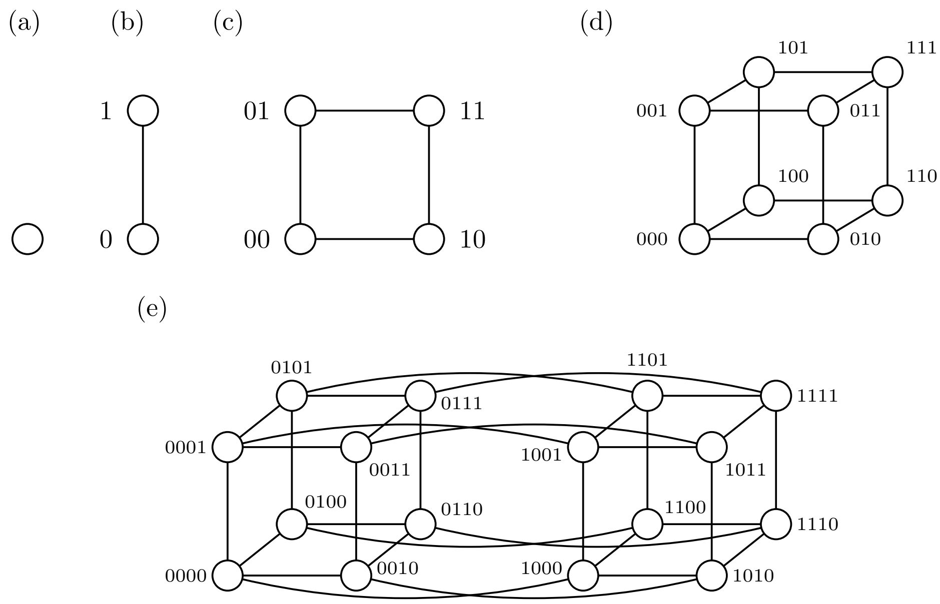
3.2 Algorytmy dla modelu PRAM

3.2.1 Sortowanie przez przestawianie nieparzysto-parzyste

```
procedure sort_nieparzyste_parzyste;
begin
  for i := 1 to ⌈n/2⌉ do
    parfor P_j, j = 1, 3, ..., 2⌊n/2⌋ − 1 do
      if A[j] > A[j + 1] then
        A[j] ⇔ A[j + 1]
      end if;
    end for;
    parfor P_j, j = 2, 4, ..., 2⌊(n−1)/2⌋ do
      if A[j] > A[j + 1] then
        A[j] ⇔ A[j + 1]
      end if;
    end for;
  end for;
end;
```



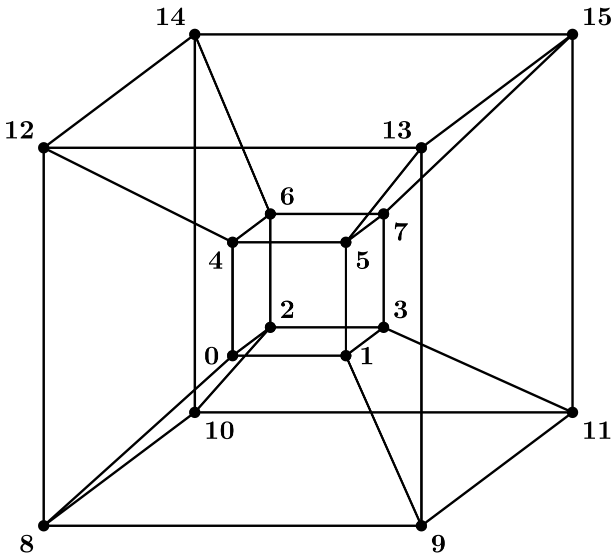
Rysunek 11: Dwuwymiarowa siatka 4×4 (a); dwuwymiarowy torus 4×4 (b) oraz trójwymiarowy torus $3 \times 3 \times 3$ (c)



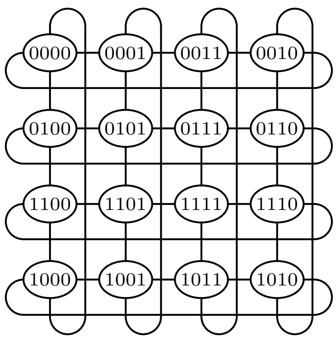
Rysunek 12: Kostka zerowymiarowa (a); jednowymiarowa (b); dwuwymiarowa (c); trójwymiarowa (d); czterowymiarowa (e)

3.2.2 Znajdowanie minimum w czasie $O(\log n)$, $n = 2^r$

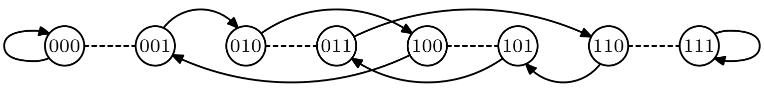
$m := a[1]$;



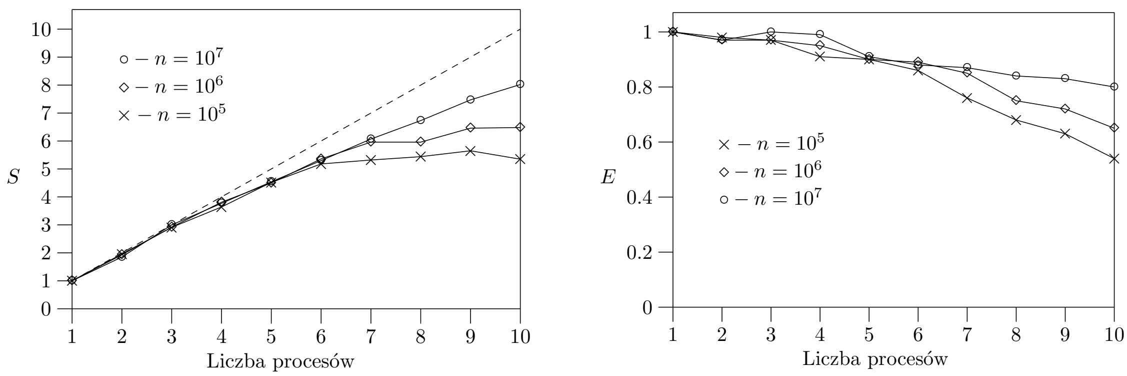
Rysunek 13: Kostka czterowymiarowa (wariant 2)



Rysunek 14: Kostka czterowymiarowa (wariant 3)



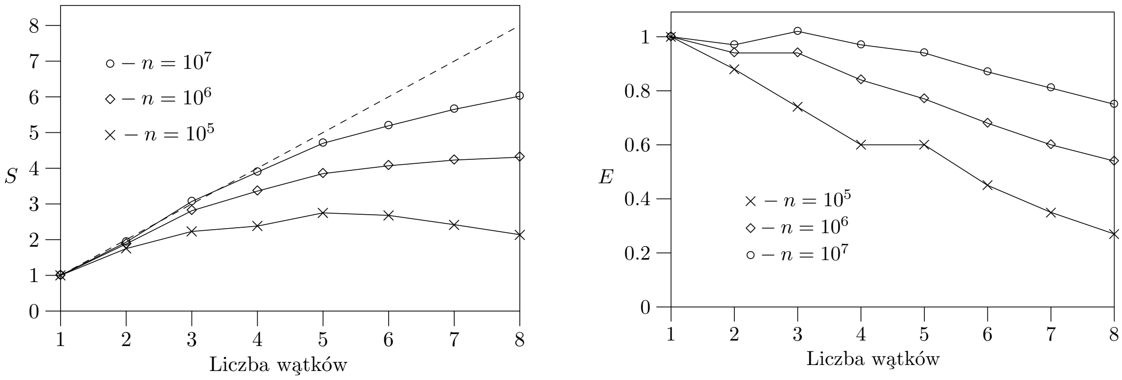
Rysunek 15: Sieć doskonale przetasowana ($m = 3$)



Rysunek 16: Przyspieszenia S oraz efektywności E w funkcji liczby procesów dla programu MPI wyznaczania liczb pierwszych; linią kreskowaną przedstawiono maksymalne przyspieszenie równe liczbie procesów

Tabela 1: Wybrane parametry charakteryzujące topologie połączeń procesorów

Rodzaj sieci	Średnica	Maks. stopień	Szerokość połowienia	Spójność krawędz.	Koszt
Sieć pełna	1	$p-1$	$p^2/4$	$p-1$	$p(p-1)/2$
Siatka jednowym.	$p-1$	2	1	1	$p-1$
Pierścień	$\lfloor p/2 \rfloor$	2	2	2	p
Siatka dwuwym.	$2(\sqrt{p}-1)$	4	$\sqrt{p}$	2	$2(p-\sqrt{p})$
Torus dwuwym.	$2\lfloor \sqrt{p}/2 \rfloor$	4	$2\sqrt{p}$	4	$2p$
Kostka	$\log p$	$\log p$	$p/2$	$\log p$	$(p \log p)/2$
Sieć id. przetas.	$2\log p - 1$	3	$p/2$	1	$3p/2$



Rysunek 17: Przyspieszenia S oraz efektywności E w funkcji liczby procesów dla programu OpenMP wyznaczania liczb pierwszych; linią kreskowaną przedstawiono maksymalne przyspieszenie równe liczbie wątków

Tabela 2: Części sekwencyjne f programów wyznaczania liczb pierwszych obliczone na podstawie przyspieszeń z rys. 16 oraz 17

	Program MPI				Program OpenMP			
	$n=10^5$		$n=10^7$		$n=10^5$		$n=10^7$	
p	S	f	S	f	S	f	S	f
2	1.96	0.020	1.93	0.036	1.75	0.143	1.93	0.036
3	2.91	0.015	2.99	0.002	2.23	0.173	3.00	0.000
4	3.64	0.033	3.94	0.005	2.38	0.227	3.89	0.009
5	4.52	0.027	4.54	0.025	2.75	0.205	4.70	0.016
6	5.18	0.032	5.30	0.026	2.68	0.248	5.19	0.031
7	5.32	0.053	6.07	0.026	2.42	0.315	5.65	0.040
8	5.44	0.067	6.73	0.027	2.14	0.391	6.02	0.047
9	5.65	0.074	7.47	0.026				
10	5.35	0.097	8.02	0.027				

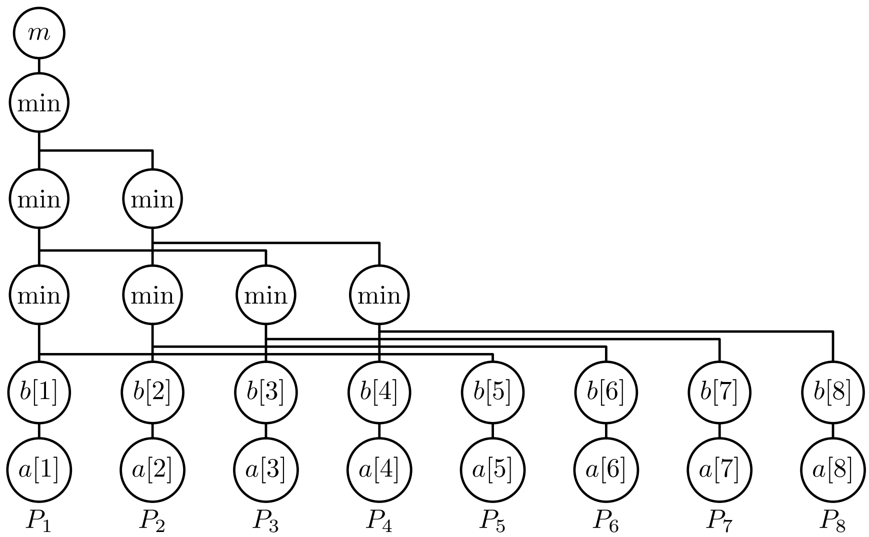
for $i := 2$ **to** n **do**
  if $a[i] < m$ **then** $m := a[i]$;
end for;

Dane wejściowe: Tablica $a[1..n]$ umieszczona w pamięci wspólnej modelu PRAM o n procesorach. Zmienne lokalne (w pamięci lokalnej każdego procesora) przechowujące rozmiar n oraz numer procesora i.

Dane pomocnicze: Tablica $b[1..n]$ umieszczona w pamięci wspólnej. Zmienne lokalne j i k

Dane wyjściowe: Wartość $\min\{a[i]\}$, dla $i = 1, 2, \ldots, n$, umieszczona w zmiennej m pamięci wspólnej.

```
begin
    parfor P_i, 1 ⩽ i ⩽ n do
        b[i] := a[i]; {kopiowanie tablicy a do tablicy pomocniczej b}
        k := n;
    end for;
    for j := 1 to log n do
        parfor P_i, 1 ⩽ i ⩽ k/2 do
            if b[i] > b[i + k/2] then
                b[i] := b[i + k/2];
            end if;
        end for;
        k := k/2;
    end for;
    if i = 1 then m := b[1]; end if;
end;
```



Rysunek 18: Ilustracja działania algorytmu wyznaczania elementu minimalnego ($n = 8$)

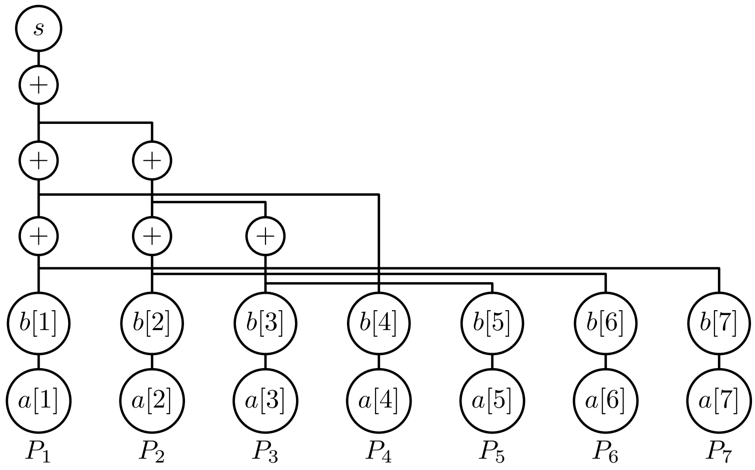
3.2.3 Znajdowanie sumy elementów w czasie $O(\log n)$, n dowolne

Dane wejściowe: Tablica $a[1..n]$ umieszczona w pamięci wspólnej modelu PRAM o n procesorach. Zmienne lokalne przechowujące rozmiar n oraz numer procesora i.

Dane pomocnicze: Tablica $b[1..n]$ umieszczona w pamięci wspólnej. Zmienne lokalne j i k.

Dane wyjściowe: Wartość $\sum_{i=1}^{n} a[i]$ umieszczona w zmiennej s pamięci wspólnej.

```
begin
    parfor P_i, 1 ⩽ i ⩽ n do
        b[i] := a[i]; {kopiowanie tablicy a do tablicy pomocniczej b}
        k := n;
    end for;
    for j := 1 to ⌈log n⌉ do
        parfor P_i, 1 ⩽ i ⩽ ⌊k/2⌋ do
            b[i] := b[i] + b[k + 1 − i];
        end for;
        k := ⌈k/2⌉;
    end for;
    if i = 1 then s := b[1]; end if;
end;
```



Rysunek 19: Ilustracja działania algorytmu wyznaczania sumy elementów ($n = 7$)

3.2.4 Znajdowanie sumy elementów za pomocą p procesorów

Dane wejściowe: Tablica $a[1..n]$ umieszczona w pamięci wspólnej modelu PRAM o p, $1 < p \leqslant n$, procesorach. Zmienne lokalne przechowujące rozmiar n, liczbę procesorów p oraz numer procesora i.

Dane pomocnicze: Tablica $b[1..p]$ umieszczona w pamięci wspólnej. Zmienne lokalne g oraz j.

Dane wyjściowe: Wartość $\sum_{i=1}^{n} a[i]$ umieszczona w zmiennej s pamięci wspólnej.

begin
    parfor P_i, $1 \leqslant i \leqslant p$ **do**
        $g := \lceil \frac{n}{p} \rceil (i-1) + 1$;
        $b[i] := a[g]$;
        for $j := 1$ **to** $\lceil \frac{n}{p} \rceil - 1$ **do** {sumowanie w grupach}
            if $g + j \leqslant n$ **then** {ostatnia grupa może być niepełna}
                $b[i] := b[i] + a[g+j]$;
            end if;
        end for;
    end for;
    Sumowanie $b[1], b[2], \ldots, b[p]$ w czasie $O(\log p)$ za pomocą poprzedniego algorytmu;
end;

Złożoność czasową, osiągane przyspieszenia oraz efektywności przedstawiają rys. 20a, b i c.

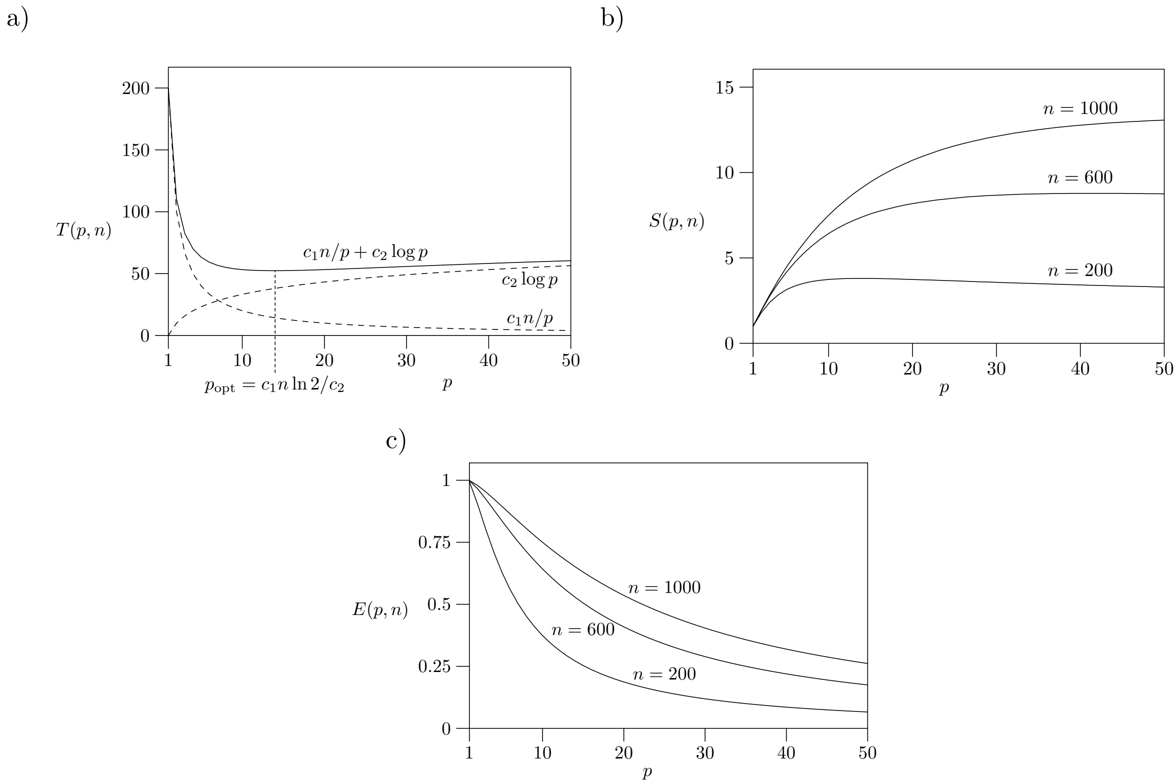
3.2.5 Problem sum prefiksowych

Dane wejściowe: Tablica $x[1..n]$ umieszczona w pamięci wspólnej modelu PRAM o n procesorach. Zmienne lokalne przechowujące rozmiar n oraz numer procesora i.

Dane pomocnicze: Zmienna lokalna j.

Dane wyjściowe: Sumy prefiksowe umieszone w tablicy $s[1..n]$ w pamięci wspólnej.

begin
    parfor P_i, $1 \leqslant i \leqslant n$ **do** {inicjalizacja}
        $s[i] := x[i]$;
    end for;
    for $j := 0$ **to** $\lceil \log n \rceil - 1$ **do**
        parfor P_i, $2^j < i \leqslant n$ **do**
            $s[i] := s[i - 2^j] + s[i]$;
        end for;
    end for;
end;



Rysunek 20: a) Złożoność czasowa algorytmu znjdowania sumy elementów za pomocą p procesorów, przyjęto $c_1 = 1.0$, $c_2 = 10$, $n = 200$; b) osiągane przyspieszenia dla różnych wartości n; c) osiągane efektywności dla różnych wartości n

Tabela 3: Ilustracja działania algorytmu wyznaczania sum prefiksowych

	Inic.	$j = 0$	$j = 1$	$j = 2$
$s[1]$	3			
$s[2]$	2	$(3) + 2$		
$s[3]$	1	$(2) + 1$	$(3) + 2 + 1$	
$s[4]$	5	$(1) + 5$	$(3 + 2) + 1 + 5$	
$s[5]$	4	$(5) + 4$	$(2 + 1) + 5 + 4$	$(3) + 2 + 1 + 5 + 4$
$s[6]$	7	$(4) + 7$	$(1 + 5) + 4 + 7$	$(3 + 2) + 1 + 5 + 4 + 7$

3.2.6 Znajdowanie minimum w czasie $O(1)$

Dane wejściowe: Tablica $a[1..n]$ umieszczona w pamięci wspólnej modelu CRCW PRAM o n^2 procesorach. Zmienne lokalne przechowujące rozmiar n oraz numer procesora w postaci pary zmiennych i oraz j.

Dane pomocnicze: Tablica $t[1..n]$ umieszczona w pamięci wspólnej.

Dane wyjściowe: Wartość $\min\{a[i]\}$, dla $i = 1, 2, \ldots, n$, zwracana przez instrukcję **return**. Jeśli wartości minimalnych jest więcej, to zwracana jest wartość minimalna $a[i]$ położona najbliżej początku tablicy a.

begin
    parfor $P_{i,j}$, $i = 1$, $1 \leqslant j \leqslant n$ **do** {pracują procesory w pierwszym wierszu}
        $t[j] := 0$; {inicjalizacja tablicy t}
    end for;
    parfor $P_{i,j}$, $1 \leqslant i, j \leqslant n$ **do** {pracują wszystkie procesory}
        if $a[i] < a[j]$ **then** {$a[j]$ nie jest elementem minimalnym}
            $t[j] := 1$;
        end if; {$t[j] = 0$ oznacza, że $a[j] = \min\{a[i]\}$, dla $i = 1, 2, \ldots, n$}

9. **if** $(t[i] = 0)$ **and** $(i < j)$ **then**
10. $t[j] := 1;$
11. **end if**; {tylko dla jednego j zachodzi $t[j] = 0$}
12. **end for**;
13. **parfor** $P_{i,j}$, $i = 1$, $1 \leqslant j \leqslant n$ **do** {pracują procesory w pierwszym wierszu}
14. **if** $t[j] = 0$ **then**
15. **return** $a[j]$; {zwrócenie elementu minimalnego}
16. **end if**;
17. **end for**;
18. **end**;

Załóżmy, że algorytm przetwarza tablicę o postaci $a[1..4] = [5, -1, 2, -1]$.

$P_{i,j}$	1	2	3	4	
1	$a[1] < a[1]$	$a[1] < a[2]$	$a[1] < a[3]$	$a[1] < a[4]$	
2	$a[2] < a[1]$	$a[2] < a[2]$	$a[2] < a[3]$	$a[2] < a[4]$	
3	$a[3] < a[1]$	$a[3] < a[2]$	$a[3] < a[3]$	$a[3] < a[4]$	
4	$a[4] < a[1]$	$a[4] < a[2]$	$a[4] < a[3]$	$a[4] < a[4]$	
t	1	0	1	0	(po wyk. kroku 8.)
t	1	0	1	1	(po wyk. kroku 12.)

Rysunek 21: Ilustracja działania algorytmu

3.2.7 Sortowanie w czasie $O(\log n)$

$$w[i,j] = \begin{cases} 1, & \text{gdy } a[i] > a[j], \text{ lub gdy } a[i] = a[j] \text{ oraz } i > j, \\ 0, & \text{w przeciwnym razie.} \end{cases} \tag{1}$$

Dane wejściowe: Tablica $a[1..n]$ umieszczona w pamięci wspólnej modelu CREW PRAM o n^2 procesorach. Zmienne lokalne przechowujące rozmiar n oraz numer procesora w postaci pary zmiennych i oraz j.

Dane pomocnicze: Tablica $w[1..n, 1..n]$ umieszczona w pamięci wspólnej. Zmienne lokalne k oraz r.

Dane wyjściowe: Tablica $b[1..n]$ umieszczona w pamięci wspólnej zawierająca wartości elementów tablicy $a[1..n]$ w porządku niemalejącym.

begin
parfor $P_{i,j}$, $1 \leqslant i, j \leqslant n$ **do**
if $(a[i] > a[j])$ **or** $((a[i] = a[j])$ **and** $(i > j))$ **then**
$w[i,j] := 1;$
else
$w[i,j] := 0;$
end if;
end for;
$k := n;$
for $r := 1$ **to** $\lceil \log n \rceil$ **do** {zliczanie jedynek w wierszach tablicy w}
parfor $P_{i,j}$, $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant \lfloor k/2 \rfloor$ **do**
$w[i,j] := w[i,j] + w[i, k+1-j];$
end for;
$k := \lceil k/2 \rceil;$
end for; {wartości $w[i,1]$ wyznaczają pozycje elementów $a[i]$ w tablicy b}
parfor $P_{i,j}$, $1 \leqslant i \leqslant n$, $j = 1$ **do**
$b[w[i,1]+1] := a[i];$
end for;
end;

$P_{i,j}$	1	2	3	4
1	$a[1] \succ a[1]$	$a[1] \succ a[2]$	$a[1] \succ a[3]$	$a[1] \succ a[4]$
2	$a[2] \succ a[1]$	$a[2] \succ a[2]$	$a[2] \succ a[3]$	$a[2] \succ a[4]$
3	$a[3] \succ a[1]$	$a[3] \succ a[2]$	$a[3] \succ a[3]$	$a[3] \succ a[4]$
4	$a[4] \succ a[1]$	$a[4] \succ a[2]$	$a[4] \succ a[3]$	$a[4] \succ a[4]$

w	1	2	3	4	$\sum$
1	0	1	1	1	3
2	0	0	0	0	0
3	0	1	0	1	2
4	0	1	0	0	1

Rysunek 22: Relacje sprawdzane przez procesory oraz postać tablicy w

3.2.8 Mnożenie macierzy

Dane wejściowe: Macierze $a[1..n, 1..n]$ i $b[1..n, 1..n]$ umieszczone w pamięci wspólnej modelu CREW PRAM o n^3 procesorach. Zmienne lokalne przechowujące rozmiar n ($n = 2^r$ dla pewnego całkowitego $r > 0$) oraz numer procesora w postaci zmiennych i, j oraz k.

Dane pomocnicze: Macierz $t[1..n, 1..n, 1..n]$ umieszczona w pamięci wspólnej. Zmienna lokalna l.

Dane wyjściowe: Iloczyn macierzy $c = a \cdot b$ umieszczony w pamięci wspólnej.

```
begin
    parfor P_{i,j,k}, 1 ⩽ i, j, k ⩽ n do
        t[i, j, k] := a[i, k] * b[k, j]; {obliczanie składowych iloczynów skalarnych}
    end for;
    for l := 1 to log n do {sumowanie składowych iloczynów skalarnych}
        parfor P_{i,j,k}, 1 ⩽ i, j ⩽ n, 1 ⩽ k ⩽ n/2^l do
            t[i, j, k] := t[i, j, 2k − 1] + t[i, j, 2k];
        end for;
    end for;
    parfor P_{i,j,k}, 1 ⩽ i, j ⩽ n, k = 1 do
        c[i, j] := t[i, j, 1];
    end for;
end;
```

3.3 Algorytmy dla modelu sieciowego

3.3.1 Mnożenie macierzy przez wektor w jednowymiarowym torusie

Dane wejściowe: Podmacierze $b = a[(i-1)r + 1..ir, 1..n]$, $r = n/p$, oraz wektor $x[1..n]$, umieszczone w procesorach P_i pracujących asynchronicznie, $1 \leqslant i \leqslant p$, połączonych w pierścień. Zmienne lokalne w każdym procesorze P_i przechowujące rozmiar n oraz numer procesora w postaci zmiennej i.

Dane pomocnicze: Wektory $y[1..n/p]$ i $z[1..n]$ umieszczone w pamięci lokalnej. Zmienne lokalne j i k.

Dane wyjściowe: Iloczyn $z = ax$ umieszczony w procesorze P_1.

```
begin
    {Obliczenie wektora y = bx przez procesory P_i, 1 ⩽ i ⩽ p.}
    for k := 1 to n/p do
        y[k] := b[k, 1] * x[1];
        for j := 2 to n do
            y[k] := y[k] + b[k, j] * x[j];
        end for;
    end for;
    if i = 1 then {procesor P_1 przesyła wektor y w prawo}
        send(y, prawy);
    else {pozostałe procesory odbierają dane z procesora położonego po lewej stronie}
        receive(z[1..(i − 1)r], lewy);
        z[1..ir] := z[1..(i − 1)r] || y; {dopisanie obliczonego wektora y}
        send(z[1..ir], prawy);
    end if;
    if i = 1 then
```

```
        receive(z[1..n], lewy); {odebranie przez procesor P_1 końcowego wyniku}
    end if;
end;
```

3.3.2 Mnożenie macierzy przez macierz w dwuwymiarowym torusie

Dane wejściowe: Macierze $a[1..n, 1..n]$ i $b[1..n, 1..n]$, których elementy $a[i, j]$ i $b[i, j]$ umieszczone są w pamięciach lokalnych procesorów $P_{i,j}$. Zmienne lokalne w każdym procesorze przechowujące rozmiar n oraz numer procesora w postaci zmiennych i oraz j.

Dane pomocnicze: Zmienna lokalna k w każdym procesorze $P_{i,j}$.

Dane wyjściowe: Iloczyn macierzy $c = ab$. Elementy macierzy $c[i, j]$ umieszczone są w zmiennych lokalnych c procesorów $P_{i,j}$.

```
begin
    {Faza 1: Zmodyfikowanie dystrybucji elementów macierzy a i b w torusie.}
    for k := 2 to n do
        parfor P_{i,j}, 1 ⩽ i, j ⩽ n do
            if i ⩾ k then {przemieszczenie wierszy macierzy a cyklicznie w lewo}
                a ⇐ prawy(a);
            end if;
            if j ⩾ k then {przemieszczenie kolumn macierzy b cyklicznie w górę}
                b ⇐ dół(b);
            end if;
        end for;
    end for;
    {Faza 2: Obliczanie iloczynu macierzy a i b.}
    for k := 1 to n do
        parfor P_{i,j}, 1 ⩽ i, j ⩽ n do
            if k = 1 then
                c := a * b ;
            else
                a ⇐ prawy(a);
                b ⇐ dół(b);
                c := c + a * b ;
            end if;
        end for;
    end for;
end;
```

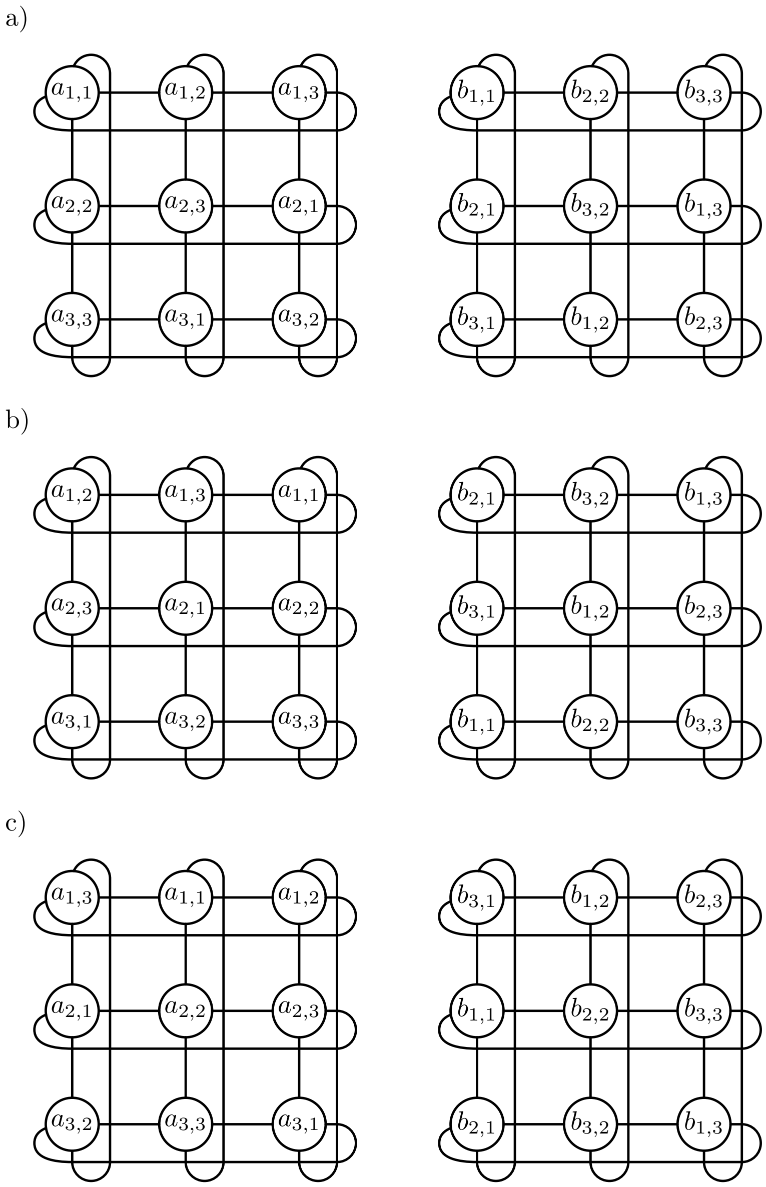
3.3.3 Problem redukcji w kostce

Dane wejściowe: Podtablice $a[1..\lceil n/p \rceil]$ zawierające fragmenty tablicy $a[1..n]$ umieszczone w pamięciach lokalnych p procesorów kostki. Zmienne lokalne przechowujące rozmiar n, liczbę procesorów p oraz numer procesora i, $0 \leqslant i < p$.

Dane pomocnicze: Zmienna lokalna *rozmiar* przechowująca rozmiar podtablicy a danego procesora oraz zmienne lokalne d, j i *pom*.

Dane wyjściowe: Wartość $\sum_{i=1}^{n} a[i]$ umieszczona w zmiennej *sum* procesora P_0.

```
begin
    parfor P_i, 0 ⩽ i < p do
        sum := a[1];
        if i < (n mod p) then
            rozmiar := ⌈n/p⌉;
        else
            rozmiar := ⌊n/p⌋;
        end if;
```

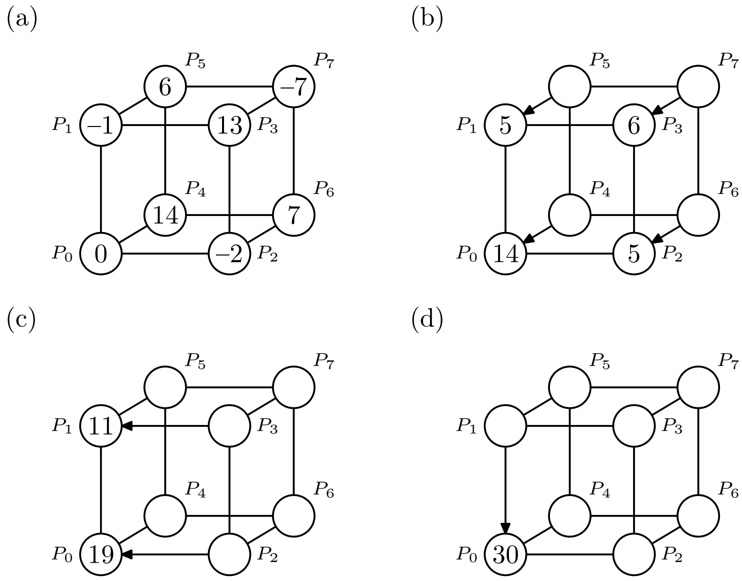


Rysunek 23: Dystrybucja elementów macierzy a oraz b pomiędzy procesorami po a) pierwszej, b) drugiej i c) trzeciej iteracji instrukcji **for** w krokach 14–24

```
        for j := 2 to ⌈n/p⌉ do {sumowanie w grupach}
            if j ⩽ rozmiar then {niektóre procesory mają mniejsze grupy}
                sum := sum + a[j];
            end if;
        end for;
    end for;
    for j := log p − 1 downto 0 do {sumowanie sum obliczonych w grupach}
        d := 2^j;
        parfor P_i, 0 ⩽ i < d do
            pom ⇐ [i + d]sum;
            sum := sum + pom;
        end for;
    end for;
end;
```

Rozgłaszanie

for $j := 0$ **to** $\log p - 1$ **do**
  $d := 2^j$;
  parfor P_i, $0 \leqslant i < d$ **do**
    $[i + d]sum \Leftarrow sum$;
  end for;
end for;



Rysunek 24: Ilustracja rozwiązywania problemu redukcji w kostce trójwymiarowej: przykładowe wartości podlegające sumowaniu (wyznaczone w wierszach 2–14) (a) oraz wartości po pierwszej (b), drugiej (c) i trzeciej (d) iteracji w wierszach 15–21 algorytmu na s. 14

3.3.4 Znajdowanie sum prefiksowych w kostce

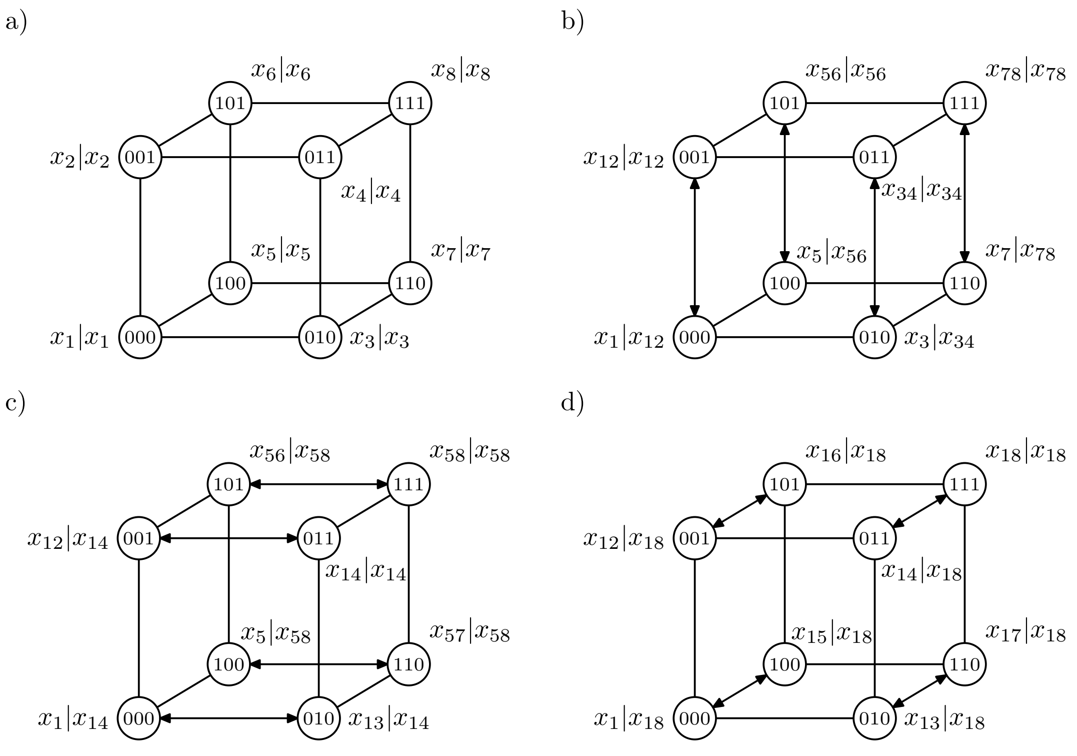
Dane wejściowe: Zmienne lokalne b w każdym procesorze kostki zainicjalizowane w taki sposób, że wartością tej zmiennej w procesorze P_0 jest element x_1, w procesorze P_1 element x_2 itd. Zmienne lokalne przechowujące liczbę procesorów p oraz numer procesora i, $0 \leqslant i < p$.

Dane pomocnicze: Zmienna lokalna j oraz m.

Dane wyjściowe: Wartości zmiennych s w procesorach P_i, $0 \leqslant i < p$, będące sumami prefiksowymi s_i, $1 \leqslant i \leqslant p$.

begin
  parfor P_i, $0 \leqslant i < p$ **do**
    $s := b$; {inicjalizacja wartości sum prefiksowych}
  end for;
  $m := i$;
  for $j := 0$ **to** $\log p - 1$ **do**
    parfor P_i, $m \bmod 2 = 0$ **do** {j-ty bit numeru procesora i jest równy 0}
      $[i + 2^j]pom \Leftarrow b$;
    end for;
    parfor P_i, $m \bmod 2 \neq 0$ **do**
      $s := s + pom$;
      $b := b + pom$;
      $[i - 2^j]b \Leftarrow b$;
    end for;

```
15.         m := m div 2;
16.     end for;
17. end;
```



Rysunek 25: Wartości zmiennych s i b w procesorach kostki a) przed wykonaniem iteracji **for** w wierszach 6–16, a następnie po wykonaniu b) pierwszego, c) drugiego i d) trzeciego przebiegu tej iteracji

3.3.5 Problem redukcji w sieci doskonale przetasowanej

Dane wejściowe: Tablica $a[1..n]$, której poszczególne elementy zostały umieszczone w zmiennych lokalnych a procesorów P_i, $0 \leqslant i < p$, $p = n$. Zmienne lokalne przechowujące liczbę procesorów p oraz numer procesora i, $0 \leqslant i < p$.

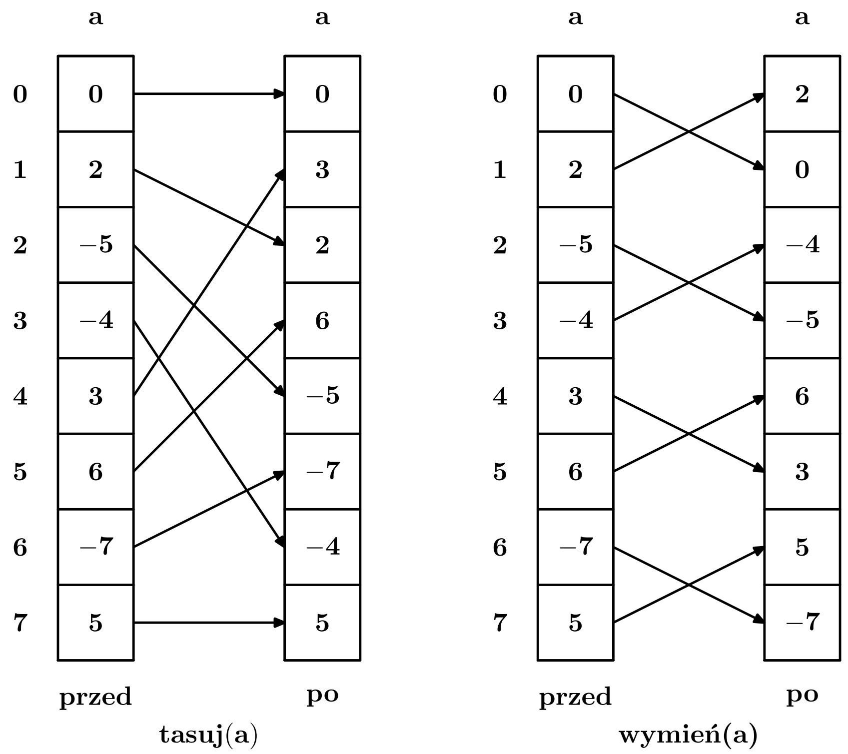
Dane pomocnicze: Zmienne lokalne b oraz j.

Dane wyjściowe: Wartość $\sum_{i=1}^{p} a[i]$ umieszczona w zmiennych a procesorów P_i, $0 \leqslant i < p$.

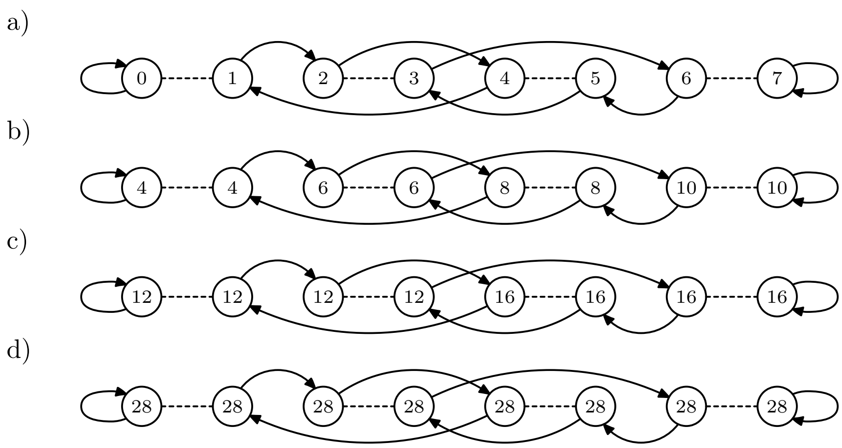
```
begin
    for j := 1 to log p do
        parfor P_i, 0 ⩽ i < p do
            tasuj(a);
            b := a;
            wymień(b);
            a := a + b;
        end for;
    end for;
end;
```

3.3.6 Problem redukcji w siatce dwuwymiarowej

```
begin
   for i := m − 1 downto 1 do
      parfor P_{j,i}, gdzie 1 ⩽ j ⩽ m do {aktywna kolumna i}
```

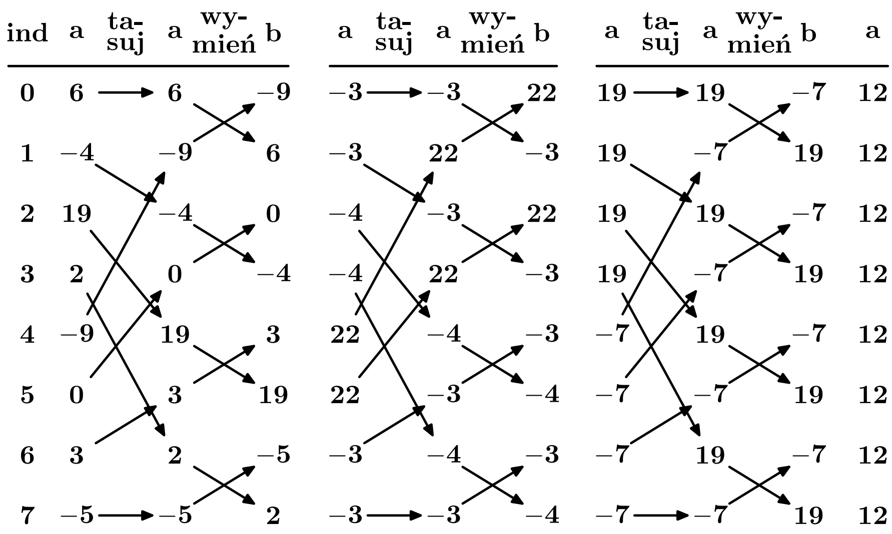


Rysunek 26: Operacje *tasuj* i *wymień*

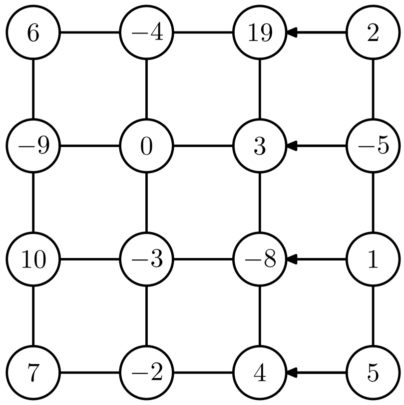


Rysunek 27: Wartości zmiennych a w procesorach sieci doskonale przetasowanej a) przed wykonaniem iteracji **for** w wierszach 2–9, a następnie po wykonaniu b) pierwszego, c) drugiego i d) trzeciego przebiegu tej iteracji ($p = 8$)

4. $\qquad t[j,\ i] \ \Leftarrow a[j,\ i+1]$;
5. $\qquad a[j,\ i] := a[j,\ i] + t[j,\ i]$;
6. $\quad$ **end for** ;
7. **end for** ;
8. **for** $i := m-1$ **downto** 1 **do**
9. $\quad$ **parfor** $P_{i,1}$ **do**
10. $\qquad t[i,\ 1] \ \Leftarrow a[i+1,\ 1]$;
11. $\qquad a[i,\ 1] := a[i,\ 1] + t[i,\ 1]$;
12. $\quad$ **end for** ;
13. **end for** ;
14. **end** ;



Rysunek 28: Redukcja w sieci doskonale przetasowanej



Rysunek 29: Redukowane (dodawane) wartości w siatce dwuwymiarowej

3.4 Algorytm dla komputera CREW MIMD

```
begin
  parfor P_i, gdzie i = 0 do -- wykonuje procesor P_0
    suma_globalna := 0;
  end for;
  parfor P_i, gdzie 0 ⩽ i ⩽ p − 1 do -- Równoległa praca p procesorów.
    suma_lokalna := 0;
    for j := i to n − 1 step p do
      suma_lokalna := suma_lokalna + a[j];
    end for;
    lock(suma_globalna);
    suma_globalna := suma_globalna + suma_lokalna;
    unlock(suma_globalna);
  end for;
end;
```

4 Komputery równoległe – klasy architektur oraz sieci połączeń

Literatura

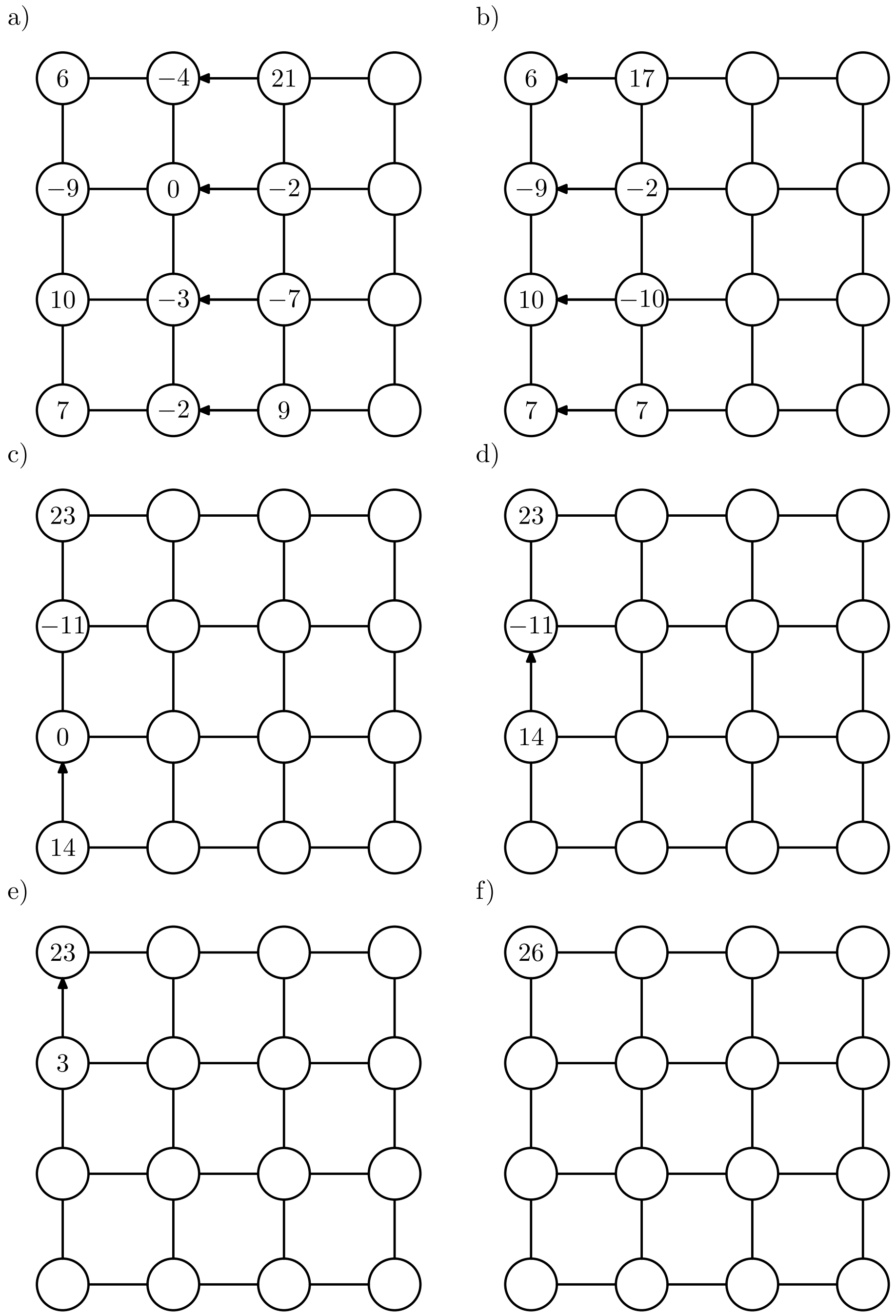
[1] Ben-Ari, M., Podstawy programowania współbieżnego i rozproszonego, WNT, Warszawa 2009.

[2] Czech, Z.J., Wprowadzenie do obliczeń równoległych, PWN, Warszawa 2013, wyd. II.

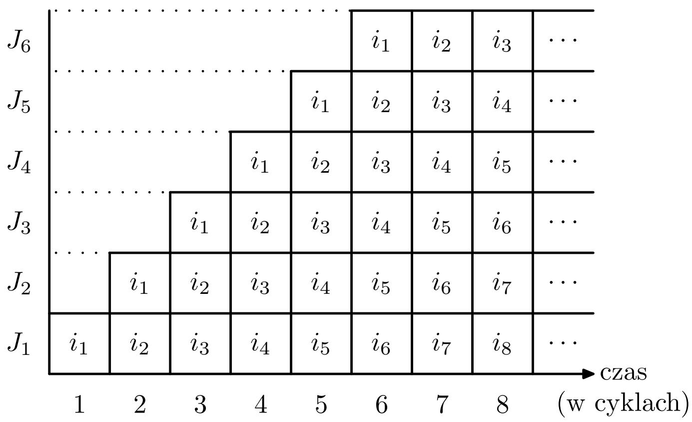
[3] Herlichy, M., Shavit, N., Sztuka programowania wieloprocesorowego, PWN, Warszawa, 2010.

[4] Huzar, Z., i in., Ada 95, Helion, Gliwice 1998.

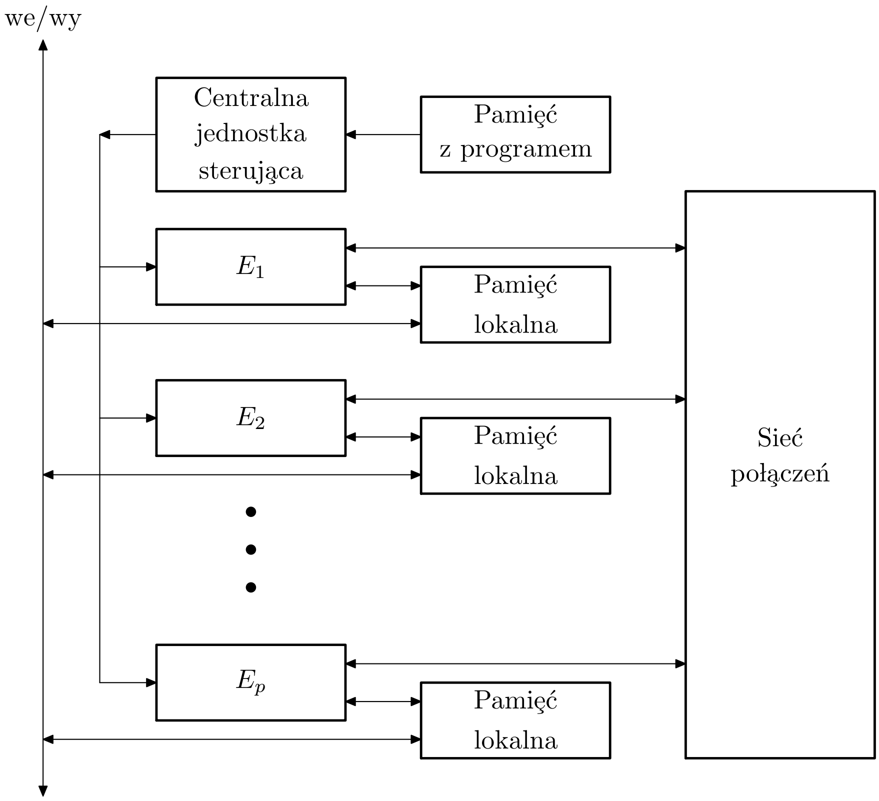
[5] Neapolitan, R., Naimipour, K., Podstawy algorytmów z przykładami w C++, Helion, Gliwice 2004 (rozdział 11 – Wprowadzenie do algorytmów równoległych).



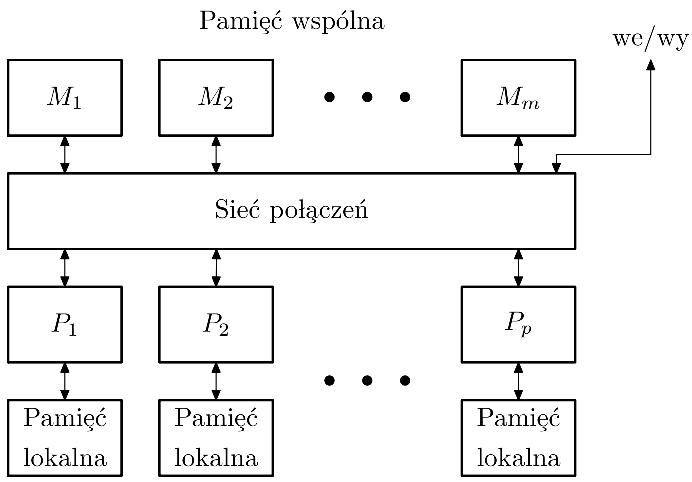
Rysunek 30: Redukcja w siatce dwuwymiarowej. Po a) pierwszym, b) drugim, c) trzecim, d) czwartym, e) piątym i f) szóstym dodawaniu



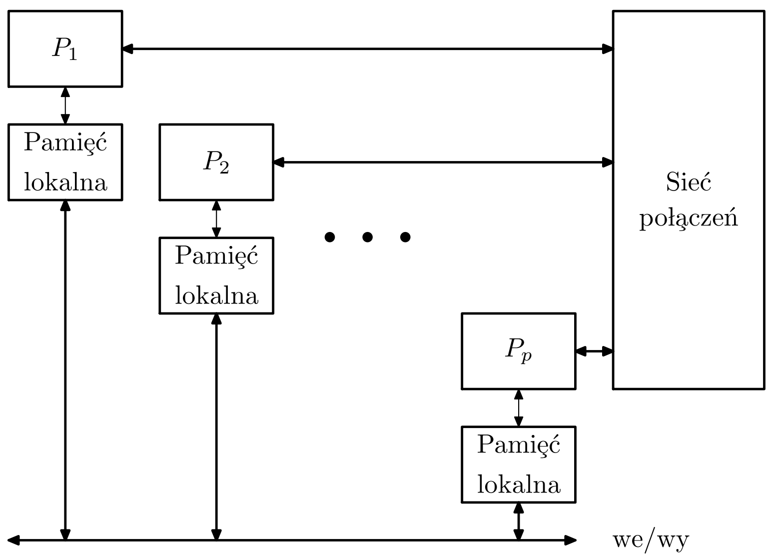
Rysunek 31: Potokowe wykonywanie strumienia instrukcji



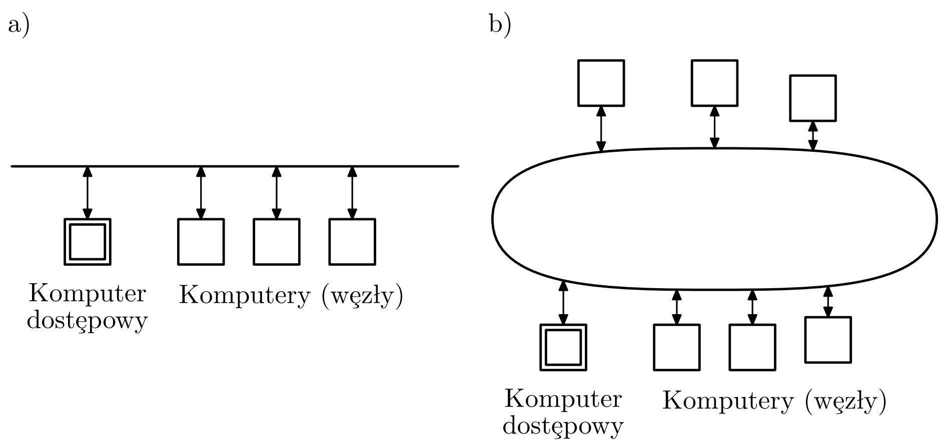
Rysunek 32: Struktura komputera macierzowego (SIMD)



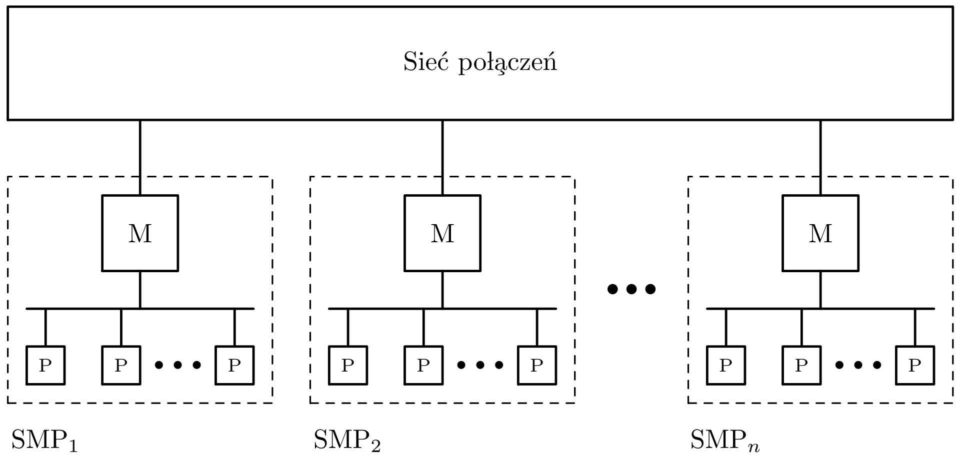
Rysunek 33: Struktura komputera wieloprocesorowego (MIMD) z pamięcią wspólną



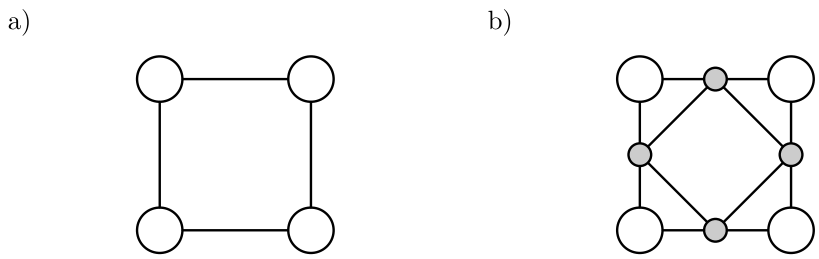
Rysunek 34: Struktura komputera wieloprocesorowego (MIMD) z pamięcią rozproszoną



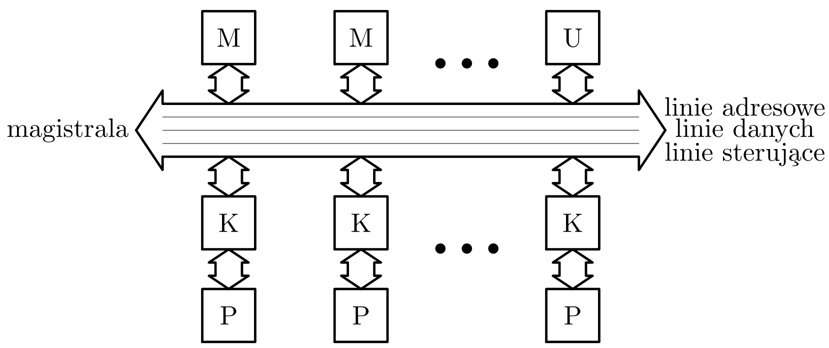
Rysunek 35: Klaster stacji roboczych a) połączonych pojedynczym przewodem, b) w postaci pierścienia



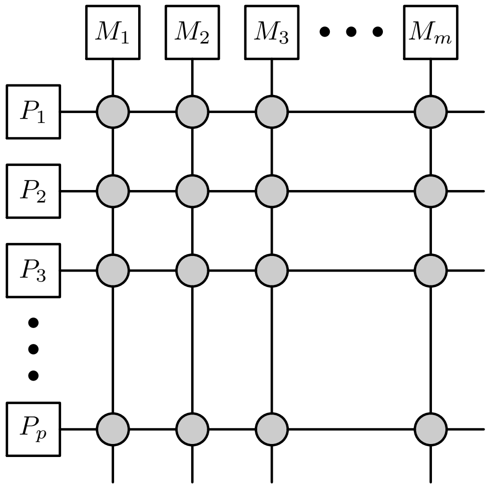
Rysunek 36: Klaster multiprocesorów symetrycznych (SMP); przez M i P oznaczono odpowiednio pamięci wspólne oraz procesory



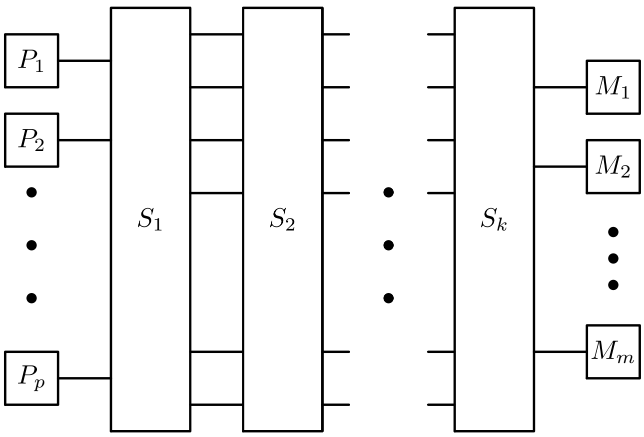
Rysunek 37: Sieć połączeń a) statyczna oraz b) dynamiczna. Węzły sieci (procesory lub bloki pamięci) oznaczono kółkami bez zacieniowania, a przełączniki kółkami zacieniowanymi



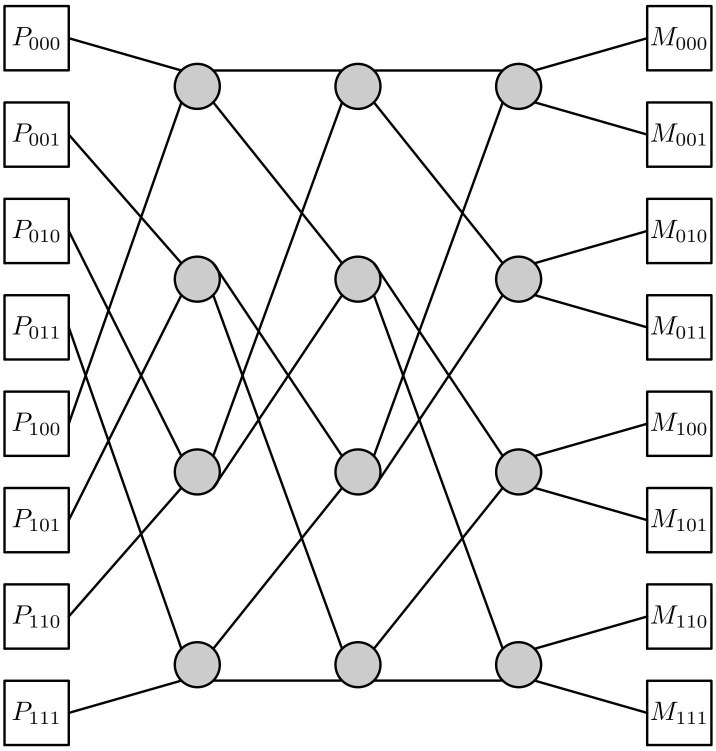
Rysunek 38: Struktura komputera z siecią połączeń opartą na magistrali (oznaczenia: P – procesory, M – bloki pamięci wspólnej, K – pamięci podręczne, U – urządzenia wej/wyj)



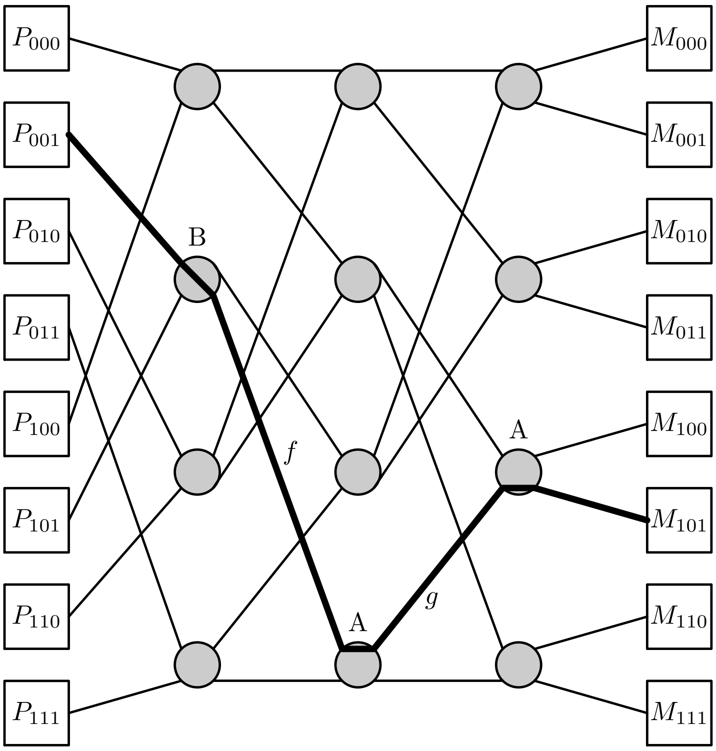
Rysunek 39: Struktura sieci z przełącznicą krzyżową



Rysunek 40: Sieć wielostopniowa: S_1, S_2, ..., S_k są stopniami sieci



Rysunek 41: Sieć omega dla $p = 8$



Rysunek 42: Stan przełączników sieci omega podczas transmisji danych pomiędzy procesorem P_{001} a blokiem pamięci M_{101}